```
# Step 1: Upload the Dataset (In Google Colab)
from google.colab import files
uploaded = files.upload()

import pandas as pd

# Load the datasets
movies = pd.read_csv("movies.csv")  # Movie data
ratings = pd.read_csv("ratings.csv")  # Ratings data

print(movies.head())
print(ratings.head())
```

Choose Files 2 files

- **movies.csv**(text/csv) - 238050 bytes, last modified: 5/13/2025 - 100% done
- **ratings.csv**(text/csv) - 1979207 bytes, last modified: 5/13/2025 - 100% done

```
Saving movies.csv to movies.csv
Saving ratings.csv to ratings.csv
   movie_id              title release_date  video_release_date  \
0         1   Toy Story (1995)  01-Jan-1995                 NaN
1         2   GoldenEye (1995)  01-Jan-1995                 NaN
2         3  Four Rooms (1995)  01-Jan-1995                 NaN
3         4  Get Shorty (1995)  01-Jan-1995                 NaN
4         5     Copycat (1995)  01-Jan-1995                 NaN

                                            IMDb_URL  unknown  Action  \
0  http://us.imdb.com/M/title-exact?Toy%20Story%2...        0       0
1  http://us.imdb.com/M/title-exact?GoldenEye%20(...        0       1
2  http://us.imdb.com/M/title-exact?Four%20Rooms%...        0       0
3  http://us.imdb.com/M/title-exact?Get%20Shorty%...        0       1
4  http://us.imdb.com/M/title-exact?Copycat%20(1995)        0       0

   Adventure  Animation  Children's  ...  Fantasy  Film-Noir  Horror  Musical  \
0          0          1           1  ...        0          0       0        0
1          1          0           0  ...        0          0       0        0
2          0          0           0  ...        0          0       0        0
3          0          0           0  ...        0          0       0        0
4          0          0           0  ...        0          0       0        0

   Mystery  Romance  Sci-Fi  Thriller  War  Western
0        0        0       0         0    0        0
1        0        0       0         1    0        0
2        0        0       0         1    0        0
3        0        0       0         0    0        0
4        0        0       0         1    0        0

[5 rows x 24 columns]
   user_id  movie_id  rating  timestamp
0      196       242       3  881250949
1      186       302       3  891717742
2       22       377       1  878887116
3      244        51       2  880606923
4      166       346       1  886397596
```

```
# Step 2: Create User-Item Interaction Matrix
interaction_matrix = ratings.pivot(index='user_id', columns='movie_id', values='rating')
interaction_matrix = interaction_matrix.fillna(0)  # Fill missing values with 0



from sklearn.decomposition import TruncatedSVD

# Apply Singular Value Decomposition (SVD)
svd = TruncatedSVD(n_components=50, random_state=42)  # Reducing to 50 components
svd_matrix = svd.fit_transform(interaction_matrix)

# Predict the ratings by multiplying the user and movie matrices
predicted_ratings = svd_matrix.dot(svd.components_)
```

```
# Create a DataFrame for the predicted ratings
predicted_df = pd.DataFrame(predicted_ratings, index=interaction_matrix.index, columns=interaction_matrix.columns

# Check the predicted ratings for a specific user
print(predicted_df.head())
```

```
movie_id      1         2         3         4         5         6     \
user_id
1         5.107651  2.517427  1.380815  2.980384  1.608721  1.750464
2         2.055177  0.018480 -0.040560  0.577979  0.095909  0.329084
3         0.306473 -0.147295 -0.110987 -0.284082  0.067498 -0.005538
4         0.107305 -0.297067  0.045807  0.189322  0.059119 -0.130069
5         4.184258  2.049577 -0.138803  1.128229  0.776645  0.070275

movie_id      7         8         9         10    ...      1673      1674  \
user_id                                           ...
1         5.131549  1.615031  3.013657  2.978584  ...  0.076903 -0.032477
2         0.791529  0.198524  2.035170  0.575685  ...  0.022124  0.013903
3         0.025827  0.577107 -0.241616  0.386882  ... -0.020658 -0.006683
4         0.101155 -0.085816 -0.070552 -0.158555  ... -0.003696  0.006513
5         1.705621  0.635517  1.067316  0.247696  ...  0.010844 -0.093746

movie_id      1675      1676      1677      1678      1679      1680  \
user_id
1        -0.033798 -0.022532  0.079559  0.001434  0.004302  0.002868
2        -0.004798 -0.003199 -0.019724  0.004629  0.013886  0.009257
3         0.056630  0.037753 -0.014570  0.014828  0.044484  0.029656
4         0.011490  0.007660  0.004451  0.006005  0.018015  0.012010
5        -0.040286 -0.026858 -0.030547 -0.002485 -0.007456 -0.004971

movie_id      1681      1682
user_id
1         0.044750  0.068175
2         0.027885 -0.018599
3        -0.002609 -0.002470
4        -0.009072 -0.031777
5        -0.017851 -0.011578

[5 rows x 1682 columns]
```

```
def recommend_movies(user_id, top_n=5):
    # Get the predicted ratings for the given user
    user_ratings = predicted_df.loc[user_id]

    # Sort movies by predicted ratings in descending order
    recommended_movie_ids = user_ratings.sort_values(ascending=False).head(top_n).index

    # Get the corresponding movie titles
    recommended_movies = movies[movies['movie_id'].isin(recommended_movie_ids)]['title']

    return recommended_movies



def recommend_by_genre(user_input, top_n=5):
    # Convert to lowercase and strip spaces
    genres_input = [genre.strip().capitalize() for genre in user_input.split(",")]

    # Validate: filter only those genres that exist as columns
    valid_genres = [genre for genre in genres_input if genre in movies.columns]

    if not valid_genres:
        return ["❌ Error: No valid genres found in input. Try genres like Action, Drama, War, etc."]

    # Filter movies that belong to at least one selected genre
    filtered_movies = movies[movies[valid_genres].sum(axis=1) > 0]

    # Merge with ratings to get average rating
    rated_movies = ratings.merge(filtered_movies, on='movie_id')
```

```
    avg_ratings = rated_movies.groupby('title')['rating'].mean().sort_values(ascending=False)

    return avg_ratings.head(top_n).index.tolist()



!pip install gradio --quiet
import gradio as gr

# Combine the user-based and genre-based recommendation functions
def gradio_interface(user_input, top_n=5):
    try:
        user_id = int(user_input)
        recommended = recommend_movies(user_id, top_n)
        return f"🎯 Recommended Movies for User {user_id}:\n\n" + "\n".join(recommended)
    except ValueError:
        # Treat as genre input
        recommended = recommend_by_genre(user_input, top_n)
        return f"🎬 Top Movies for Genre(s): {user_input}\n\n" + "\n".join(recommended)


# Create the Gradio interface
gr.Interface(
    fn=gradio_interface,
    inputs="text",
    outputs="text",
    title="🎬 Personalized Movie Recommender",
    description="Enter a User ID to get personalized movie recommendations or enter genres like 'Action, Drama' t
).launch(share=True)
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.1/54.1 MB 17.7 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.9/322.9 kB 22.0 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB 7.8 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.5/11.5 MB 91.3 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 3.9 MB/s eta 0:00:00
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 4.7 MB/s eta 0:00:00
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://18632173e57905a6a5.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from th
```